#6

Submitted to Algorithmica

PROXIMITY PROBLEMS FOR POINTS ON A RECTILINEAR PLANE WITH RECTANGULAR OBSTACLES *

SUMANTA GUHA† ICHIRO SUZUKI‡

Electrical Engineering & Computer Science Department
University of Wisconsin-Milwaukee
Milwaukee, WI 53201
USA
Email: guha@cs.uwm.edu, suzuki@cs.uwm.edu
Tel: 414-229-5810 (Guha), 414-229-3718 (Suzuki)
Fax: 414-229-6958

Abstract

We consider the following four problems for a set S of k points on a plane, equipped with the rectilinear metric and containing a set R of n disjoint rectangular obstacles (so that distance is measured by a shortest rectilinear path avoiding obstacles in R): (a) find a *closest pair* of points in S, (b) find a *nearest neighbor* for each point in S, (c) compute the rectilinear *Voronoi diagram* of S, and (d) compute a rectilinear *minimal spanning tree* of S. We describe $O((n+k)\log(n+k))$ time sequential algorithms for (a) and (b) based on *plane-sweep*, and the consideration of geometrically special types of shortest paths, so-called *z-first paths*. For (c) we present an $O((n+k)\log(n+k)\log n)$ time sequential algorithm that implements a sophisticated *divide-and-conquer* scheme with an added *extension phase*. In the extension phase of this scheme we introduce novel geometric structures, in particular so-called *z-diagrams*, and techniques associated with the Voronoi diagram. Problem (d) can be reduced to (c) and solved in $O((n+k)\log(n+k)\log n)$ time as well. All our algorithms are *near-optimal*, as well as easy to implement.

Keywords: Computational geometry, rectilinear metric, obstacles, nearest neighbors, Voronoi diagram, minimal spanning tree.

1 Introduction

A fundamental problem in computational geometry is, given a geometric space G equipped with some metric d, that of computing shortest paths in G. This leads to several proximity problems when one is, in addition, given a finite subset S of G.

We consider the case when G is the Cartesian plane, together with an obstacle set R consisting of n disjoint isothetic rectangles (i.e., with sides parallel to the coordinate axes), and the metric d (often called the *rectilinear* or *Manhattan* metric) is defined such that, if $p, q \in G$, then $d(p,q)$ is the Euclidean length of a shortest *rectilinear path* (i.e., consisting of axes-parallel segments) joining p and q that does not intersect the interior of any of the rectangular obstacles in R. Such geometric spaces arises naturally in applications such as VLSI chip design, plant and facility

*An extended abstract appeared in *Proc. 13th Conf. on the Foundations of Software Tech. and Th. Comp. Sc.*, Bombay, 1993, Springer-Verlag, 218-227.

†Supported in part by a UW-Milwaukee Graduate School Research Committee Award.

‡Supported in part by the National Science Foundation under grants CCR-9004346 and IRI-9307506, the Office of Naval Research under grant N00014-94-1-0284, and an endowed chair supported by Hitachi Ltd. at Faculty of Engineering Science, Osaka University.

JGK

layout, robot motion planning, and urban transportation. These spaces, as well as various generalizations, have been investigated extensively for efficient sequential [5, 6, 11, 13, 14, 18, 19] and parallel [2, 8] algorithms to compute shortest paths.

The problems considered in this paper are, given a plane G, containing an obstacle set R of n rectangles and equipped with a metric d, as described above, and, further, a set S of k points in $G - \cup_{r \in R} r$, to

(a) find a closest pair of points in S,

(b) find a nearest neighbor for each point in S,

(c) compute the Voronoi diagram of S, and

(d) compute a minimal spanning tree of S.

We give near-optimal sequential algorithms for all four problems (in fact, each algorithm may be sub-optimal by at most a logarithmic factor), thus resolving questions open since at least 1985 when de Rezende, Lee, and Wu [6] gave optimal algorithms to find shortest paths in such a space.

Section 2 introduces some terminology and preliminary results, as well as the notion of so-called z-first paths (where z is one of the four directions, $\pm x$ and $\pm y$), which are shortest paths of a special geometric type.

In Section 3, we describe an $O((n+k)\log(n+k))$ time algorithm that finds a closest pair in S after sweeping the plane in the $+x$ and $+y$ directions to determine shortest x-first and y-first paths.

Section 4 describes a similar algorithm with the same time bound to find all nearest neighbors in S, but in this case the plane is swept in all four directions, $\pm x$ and $\pm y$.

In Section 5, we describe the more complicated algorithm to compute the Voronoi diagram of S. Our algorithm runs in $O((n+k)\log(n+k)\log n)$ time, which is quicker, but not significantly so, than the next best Voronoi diagram algorithm that we are aware of for a similar geometric space: the algorithm of Mitchell [14, see Theorem 2] which runs in $O((n+k)\log^2(n+k))$ time. However, what may be of more interest is that, while Mitchell's algorithm uses a "continuous Dijkstra" method of propagating a "wavefront" from each point of S as a source, ours is quite different and based on a divide-and-conquer *with* an "extension phase". It should be pointed out though that Mitchell's method allows the more general class of simple polygons as obstacles, while it is not clear if our methods can be extended beyond the class of rectangular obstacles.

Considering another geometric space with obstacles, Aronov [1] achieves a time bound identical to ours for computing the Euclidean Voronoi diagram of k points in an n-sided simple polygon. His overall scheme is also divide-and-conquer with an extension phase but, our space being dissimilar, we differ significantly in the implementation of the scheme and, in fact, introduce new geometric structures and methods. In particular, we exploit the special geometry of the rectilinear plane to implement the crucial extension phase in two stages: in the first stage we compute "approximate" Voronoi extensions, so-called z-diagrams (where z is one of the four directions, $\pm x$ and $\pm y$), and then, in the next stage, use these approximations to sweep through while tracing out the boundaries of the "exact" Voronoi cells.

In Section 6, we discuss the reduction of the problem of computing a minimal spanning tree of S to that of computing the Voronoi diagram of S. This leads to an $O((n+k)\log(n+k)\log n)$ time algorithm to compute a minimal spanning tree, implying almost linear order speed-up over the minimal spanning tree algorithm of Wu, Widmayer, Schlag, and Wong [18] (which runs in $O(k\log k + n^2\log n)$ time, but allows the more general class of rectilinear convex polygons as obstacles).

We conclude in Section 7 with a discussion of the near-optimality of all our algorithms and related open questions.

Throughout, we avoid repeating proofs that have appeared in the available literature.

A-1

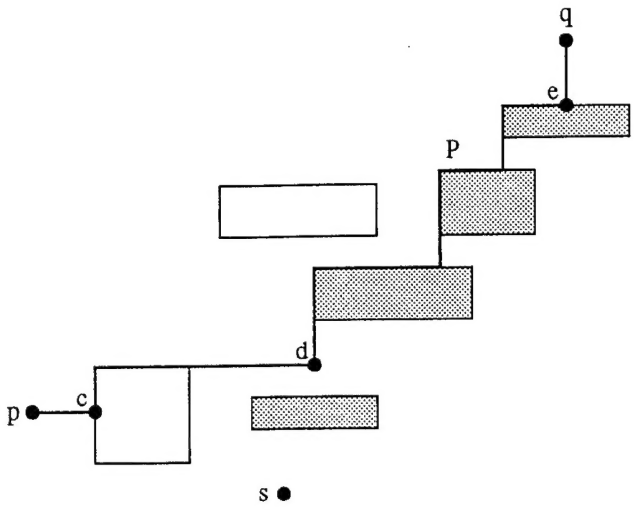

Figure 1: The shaded rectangles comprise a horizontal blockage between s and q; P is the x-first path from p to q.

2 Preliminaries

We shall henceforth assume that all paths are rectilinear and avoid intersecting the interiors of rectangles in R. A path in G from a point p to a point q that is monotone in the x direction is called an *x-path*, a path monotone in both x and y directions is called an *xy-path*, and similarly denote paths monotone in other directions ($-x, -y$, etc.).

A *horizontal blockage*, if one exists, between a pair of points s and q is a subset of rectangles of R dispositioned as indicated in Figure 1 (in which case, we also say s is *horizontally blocked* from q). More precisely (following [6]):

Assume $s = (s_x, s_y)$ and $q = (q_x, q_y)$ and, without loss of generality, that $s_x < q_x$ and $s_y < q_y$. For a rectangle $r \in R$, let $min_x(r)$ and $max_x(r)$ denote the smaller and larger of the x-coordinates of the vertical edges of r, respectively. Similarly, define $min_y(r)$ and $max_y(r)$.

Given a sequence $H = \{r_1, r_2, \ldots, r_h\}$ of rectangles of R, we say that H is a *horizontal blockage* from s to q if

1. $s_y < min_y(r_1)$,
2. $max_y(r_i) < min_y(r_{i+1}) \ \forall i = 1, 2, \ldots, h-1$,
3. $max_y(r_h) < q_y$,
4. $max_x(r_1) > s_x > min_x(r_1)$,
5. $max_x(r_i) > min_x(r_{i+1}) \ \forall i = 1, 2, \ldots, h-1$, and
6. $max_x(r_h) > q_x > min_x(r_h)$.

Similarly define *vertical blockage*. We refer to [6] for a proof of the following:

Proposition 1 ([6]) *Between two points p and q there cannot be both a horizontal as well as a vertical blockage (there may be neither). If there is no horizontal blockage and p is left of q (i.e., the x-coordinate of $p \leq$ the x-coordinate of q), then any shortest path from p to q is an*

x-path. If there is no horizontal blockage and p is right of q, then any shortest path from p to q is a $(-x)$-path.

Similar results hold if there is no vertical blockage. □

Shortest paths joining two points are never unique (except in the trivial case when the shortest path consists of only one segment), but in the following proposition we introduce and define a special type of shortest paths, *z-first paths*, with a geometric property that almost always makes them unique:

Proposition 2 *If there is no horizontal blockage between p and q and p is left of q, then there is at least one shortest x-path from p to q, call such an* x-first path from p to q, *that always proceeds in the x direction unless it would either enter inside a rectangle, or enter a region between each point of which and q there does* not exist *an x-path (i.e., each point of that region is either horizontally blocked from q or lies to the right of q). Thus, the x-first path from p to q makes a turn in a $\pm y$ direction only when it either hits a rectangle, or "risks" loosing x-monotonicity. See Figure 1.*

Similar results hold if p is right of q, or if there is no vertical blockage. In particular, we have similar definitions for z-first paths, where $z = -x$ or $\pm y$.

Proof. A precise *iterative* procedure for drawing an x-first path from p to q is as follows:

Say the source $p = (p_x, p_y)$ and destination $q = (q_x, q_y)$. Draw the path from p in the x direction until (whichever comes first)

1. it hits, at point c, the left edge of some rectangle, or
2. it reaches a point $d = (\alpha, p_y)$ such that, for sufficiently small $\epsilon > 0$, $(\alpha+\epsilon, p_y)$ is horizontally blocked from q, or
3. it reaches the point $e = (q_x, p_y)$.

In case 1, say the corners of the edge on which c lies are a and b. Draw the path vertically from c to a if $d(c,a) + d(a,q) < d(c,b) + d(b,q)$, or to b if $d(c,a) + d(a,q) > d(c,b) + d(b,q)$, or, arbitrarily, to either a or b if $d(c,a) + d(a,q) = d(c,b) + d(b,q)$ (this is exactly the case when the x-first path is not unique). See Figure 1. *Repeat* the drawing procedure with the current endpoint (either a or b) of the path as the new source.

In case 2, let the rectangle r, from a horizontal blockage between $(\alpha + \epsilon, p_y)$ and q, be the one that is vertically adjacent to $(\alpha + \epsilon, p_y)$, for sufficiently small $\epsilon > 0$. Then, clearly, the left edge of r has corners a and b with x-coordinate equal to α. Draw the path vertically from d to the more distant of a and b. See Figure 1. *Repeat* the drawing procedure with the current endpoint (either a or b) of the path as the new source.

In case 3, draw the path vertically from e to q. This is possible as, by case 2, we never reach a point that is horizontally blocked from q. See Figure 1. *Exit.*

Clearly, this procedure completes, after a finite number of turns, an x-path P from p to q. It remains to show that P, which we call the x-first path from p to q, is indeed shortest. We shall prove this by induction on the number of segments of P.

Starting the induction is trivial. Assume inductively then that P has $n(> 1)$ segments, and that all x-first paths with no more than $n-1$ segments are shortest. If possible let Q be a path from p to q that is shorter than P. Then P and Q are disposed as in either Figure 2(a) or 2(b) (we may assume without loss that they do not intersect).

Assume first that they are disposed as in Figure 2(a). Let m be the other end of the first segment of P leaving p (this segment is depicted as horizontal in Figure 2, but it may as well be vertical). From m draw the xy-path R with y-preferred (i.e., the path which, whenever it can

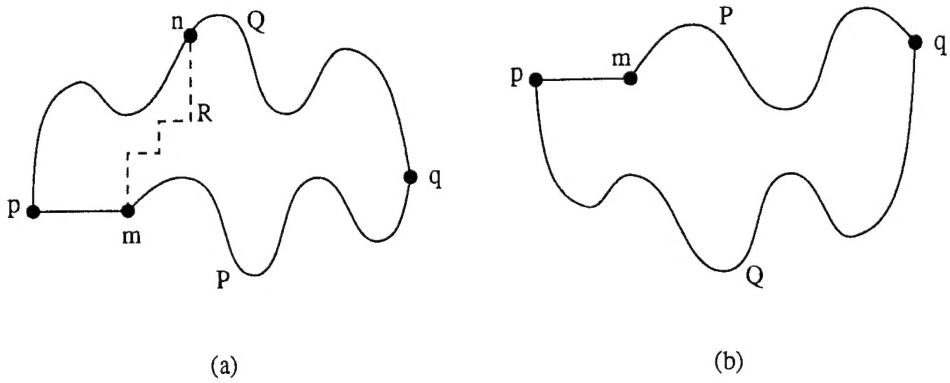

Figure 2: Illustration for the proof of Proposition 2.

go in either the x or y direction without intersecting the interior of a rectangle, chooses the y direction, see [6]). As m is not horizontally blocked from q, R intersects Q at some interior point n. Denoting, for example, the distance along P from p to q by $|P(p,q)|$, we have by assumption

$$\begin{aligned} |Q(p,q)| &< |P(p,q)| \\ \Rightarrow |Q(p,n)| + |Q(n,q)| &< |P(p,m)| + |P(m,q)| \end{aligned} \tag{1}$$

As $P(p,m) \cup R(m,n)$ is an xy-path, it is a shortest path from p to n, so that

$$|P(p,m)| + |R(m,n)| \leq |Q(p,n)| \tag{2}$$

From (1) and (2) we have

$$\begin{aligned} |P(p,m)| + |R(m,n)| + |Q(n,q)| &< |P(p,m)| + |P(m,q)| \\ \Rightarrow |R(m,n)| + |Q(n,q)| &< |P(m,q)|, \end{aligned}$$

contradicting the inductive hypothesis as $P(m,q)$ is an x-first path with $n-1$ segments, and proving Q cannot exist. If P and Q are disposed as in Figure 2(b), an exactly similar argument holds after drawing R from m as the xy-path with $(-y)$-preferred. □

Comment. The notion of, for example, *x-preferred paths* in [6] is different from ours of x-first paths in that it does not characterize shortest paths between given pairs of points.

The following *separator* result can be proved following [2] (where, in fact, an n-processor $\log n$-time PRAM algorithm is given) with straightforward modifications, also see Figure 3:

Proposition 3 ([2]) *In $O(n \log n)$ time one can describe an $x(-y)$-path P (after re-orienting the coordinate axes if necessary), comprising $O(n)$ segments and unbounded in both directions with the first and last segments being* vertical *(P is imagined to start from a point at infinity in the y direction), such that there are at least $\frac{1}{8}n$ rectangles of R on either side of P.* □

3 Finding a Closest Pair

It is worth noting at the outset that the Bentley-Shamos [3] divide-and-conquer scheme cannot be directly applied, as it is no longer true that a "circle" of radius r in the d-metric around a

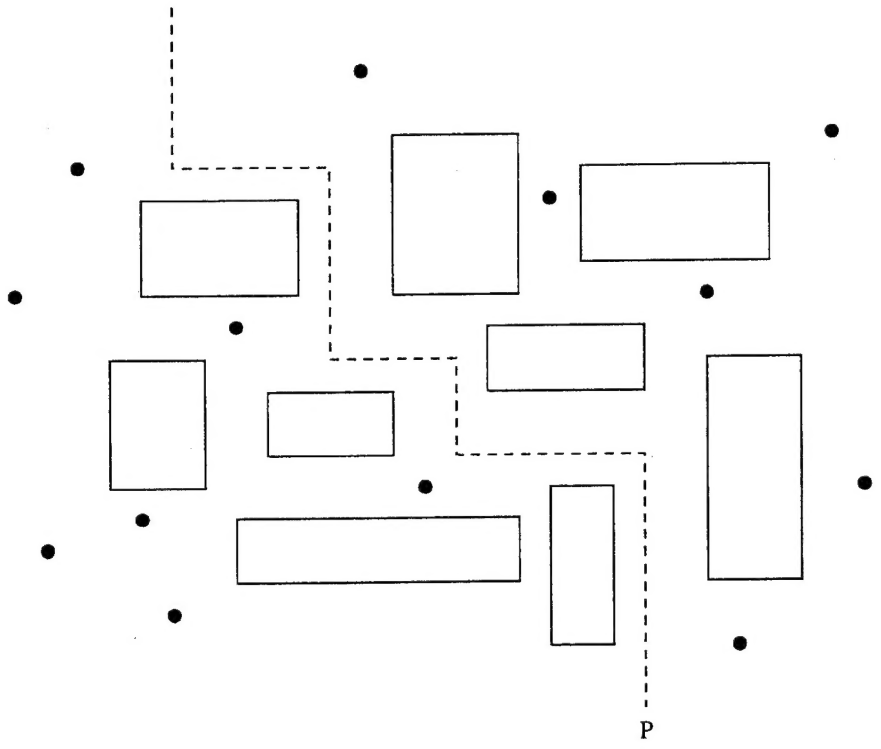

Figure 3: P is a separator as in Proposition 3.

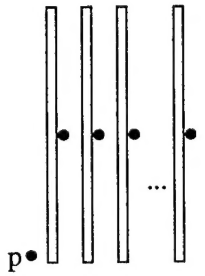

Figure 4: Each unlabeled point is at distance less than r from p, but at distance greater than r from each other.

point p can contain at most $O(1)$ points each of which is at distance at least r from the other. See Figure 4. Instead, we exploit properties of the rectilinear metric for a very different method based on plane-sweep. Our idea is, while sweeping in the x direction for instance, to gather information about shortest x-first paths. It should be pointed out that the algorithm in [6] for the single-source shortest path problem in a similar setting uses plane-sweep as well, though, for our purposes, we need to maintain considerably more information and data structures through the sweep.

Let C_l denote the set of corners on the *left* edges of rectangles of R. Further, as a simplifying device, add to S a new point $(-\infty, 0)$ (we shall, in fact, assume $(-\infty, 0)$ to be a finite point sufficiently far left of any existing member of $S \cup C_l$).

Suppose L_x is a vertical sweep-line that begins to scan in the x direction starting from a position just right of $(-\infty, 0)$. Maintain the status of L_x in a height-balanced search tree (e.g., a red-black tree) T_x, such that, if at time t the sweep-line lies on $L_x(t)$, the status tree at that instant, denoted $T_x(t)$, represents an increasing set of points $\{-\infty = a_0, a_1, \ldots, a_{l_t} = \infty\}$ on $L_x(t)$ ($L_x(t)$ is, of course, imagined to be a copy of the real line by a projection of the y axis), together with a *label* B_i for each point $a_i, 1 \leq i \leq l_t$, such that

1. For each i, either B_i is of the form (p, c), where $p \in S$ and $c \in S \cup C_l$, both p and c lying left of $L_x(t)$; or, B_i is of the form r, where $r \in R$.

2. If $B_i = (p, c)$ then, for each point q in the open interval (a_{i-1}, a_i) on $L_x(t)$, p is the closest to q of the points of S from which there is an x-path to q (clearly, all such points must lie left of $L_x(t)$, and $(-\infty, 0)$ is one such); further, there is a unique x-first path, say P, from p to q, and the last point of $S \cup C_l$ at which P turns as it moves from p to q is c (counting p itself as the first and possibly only such turn point). Call c the *x-anchor* of q w.r.t. p.

3. If $B_i = r$ then either (a_{i-1}, a_i) is the left edge of r (minus endpoints) or lies in the interior of r.

4. The sorted set $\{a_0, a_1, \ldots, a_{l_t}\}$ is minimal with respect to the labeling, i.e. no two adjacent a_i share the same label.

See Figure 5 . We shall often refer to an open interval as being associated with the same label as its right endpoint. Initially, of course, $T_x(0)$ contains only the points $-\infty$ and ∞, with the point ∞ having label $((-\infty, 0), (-\infty, 0))$.

In addition to the status tree T_x, maintain an array D, indexed by $S \cup C_l$, such that, at time t,

- if q is left of $L_x(t)$ (i.e., if L_x has already swept over q), the entry $D[q]$ contains the name of the point which is closest to q amongst points of S, different from q, from which there is an x-path to q, as well as the distance of that point from q, and

- if $L_x(t)$ is strictly left of q, then $D[q] = \infty$, denoting 'no information available' (as a special case, mark $D[(-\infty, 0)] = \infty$ throughout).

As L_x moves rightwards the following three types of events occur at various times:

1. $L_x(t)$ touches a point of S, called a *point* event.

2. $L_x(t)$ touches the left side of a rectangle, called a *left* event.

3. $L_x(t)$ touches the right side of a rectangle, called a *right* event.

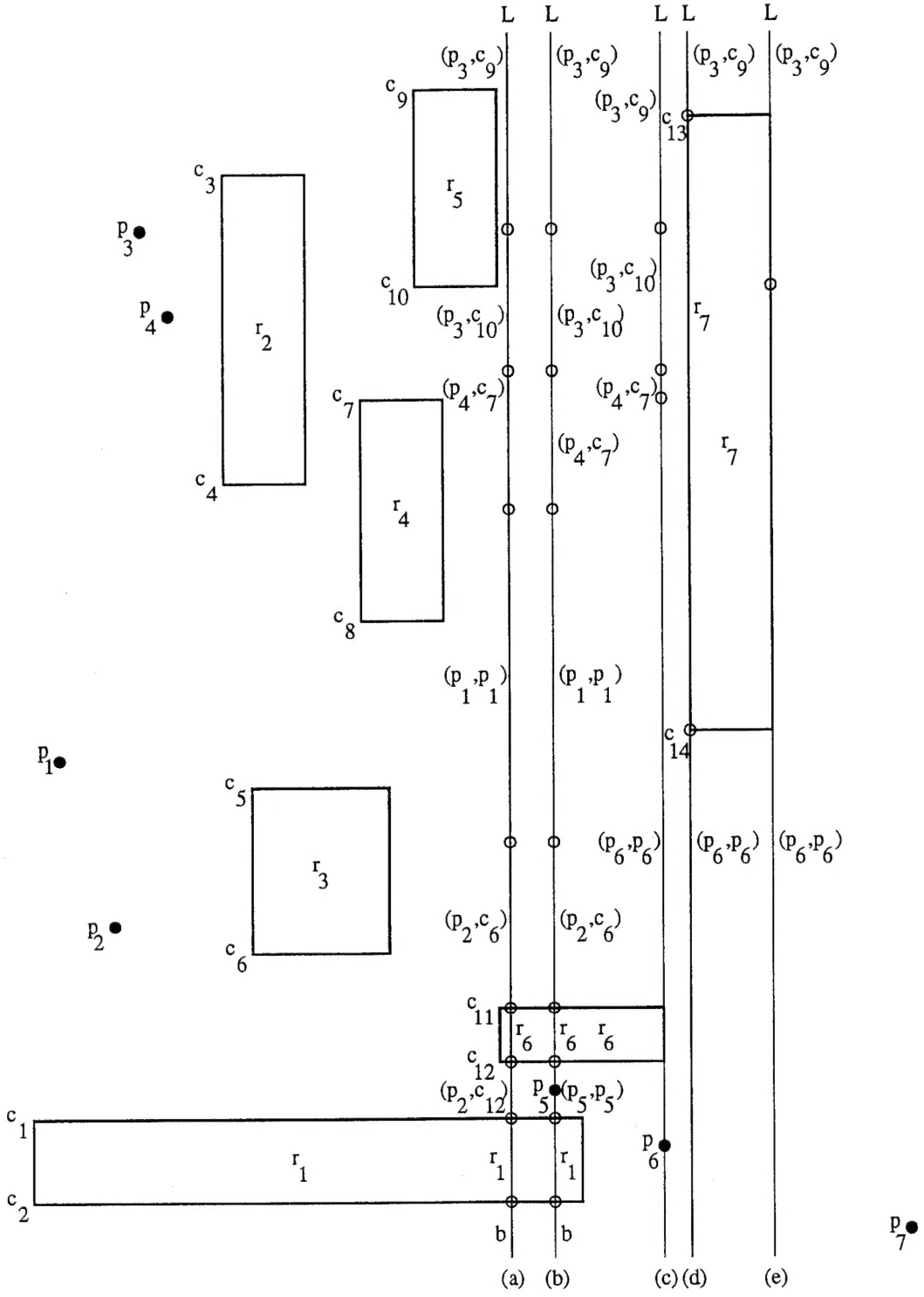

Figure 5: The status of the sweep line is shown at five positions: (a) just before the point event at p_5, (b) after updating for the point event at p_5, (c) after updating for the point event at p_6, (d) after updating for the left event at the left edge of r_7, and (e) after updating for the right event at the right edge of r_7. The solid circles are the points p_i of S, the hollow circles are points of the status tree, the r_i are the rectangles, and the c_i are corners on the left edges of rectangles. The labels are shown next to the intervals, and label $b = ((-\infty, 0), c_2)$.

We assume, for simplicity, that no two events happen at the same time (i.e., points of S and vertical edges of rectangles of R all have distinct x-coordinates). Removing this restriction is a minor technicality. Sort the points of S and the vertical edges of rectangles of R by x-coordinate, in $O((n+k)\log(n+k))$ time, to obtain the schedule of events. The following is not hard to prove:

Lemma 1 *$T_x(t)$ does not change between successive events.*

Proof. Follows by elementary geometric arguments, given the rectilinearity of the metric. □

Next, we describe the procedures to update D and T_x at each type of event.

Point event: At a point event at time t, $L_x(t)$ touches a point of S, say q. Assume the previous event took place at time t'.

By searching $T_x(t')$ locate q (or, more accurately, its y-coordinate) in the array $a_0, \ldots, a_{l_{t'}}$. Say $a_{i-1} < q \leq a_i$.

If the label $B_i = (p, c)$ (it cannot be an $r \in R$), then p is a nearest one to q (we have a choice when $q = a_i$) amongst points of S, different from q, from which there is an x-path to q; and c is the x-anchor of q w.r.t. p. Further, the distance along a shortest x-path from p to q can be determined as the sum of the distance of c from p (which can be read from array D), the length of the perpendicular from c to $L_x(t)$, and the distance of q from the base $\overline{c}$ of this perpendicular (which lies in (a_{i-1}, a_i)). In fact, the function $d_i : [a_{i-1}, a_i] \rightarrow \Re$, the reals, giving distances of points from p along x-paths, attains a minimum at $\overline{c}$ and linearly increases with gradient one on either side of that base.

Updating D: Update the entry $D[q]$ with the name of p and its distance from q.

Updating T_x: To update $T_x(t')$, a new interval with label (q, q) must be created that contains q and extends on either side of q to contain points of $L_x(t)$ that are now closer to q than they are to points of S strictly left of $L_x(t)$, considering, of course, only distances measured along x-paths. Determining the extent of this new interval is straight-forward: simply proceed in either direction from q along $L_x(t)$ comparing the distance from q to the distance from the currently known nearest point of S (as given by the distance function d_j of the interval one is currently inside), and stopping only if the two distances become equal or the side of a rectangle of R is reached. This will, of course, result in the partial or full deletion of some vertices of $T_x(t')$. Updating $T_x(t')$ consists, therefore, of inserting the two endpoints of the new interval and deleting old vertices that lie inside the new interval. The upper endpoint of the new interval will have label (q, q), while the lower endpoint will have the label of the interval of $T_x(t')$ in which it lies.

See Figure 5.

Left event: At a left event at time t, $L_x(t)$ touches the left side, say $[r_1, r_2]$, of some rectangle $r \in R$, and we imagine r to become active as an obstacle intersecting $L_x(t)$.

Updating D: Update $D[r_1]$ and $D[r_2]$ by searching $T_x(t')$ to locate r_1 and r_2 in $a_0, \ldots, a_{l_{t'}}$, and performing appropriate distance computations.

Updating T_x: Updating $T_x(t')$ consists firstly of inserting r_1 and r_2 as new vertices, and then deleting all vertices that lie in $[r_1, r_2]$.

Label r_2 with r.

If the label of the vertex following r_2 is (p, c) such that the base of the perpendicular from c to $L_x(t)$ lies below r_2, then relabel this vertex with (p, r_2).

If r_1 lies in an interval of $T_x(t')$ with label (p', c'), then label r_1 with (p', c') or (p', r_1) according as the base of the perpendicular from c' to $L_x(t)$ lies below or above r_1.

See Figure 5.

Right event: At a right event at time t, $L_x(t)$ touches the right side, say $[r_3, r_4]$, of some rectangle $r \in R$, and we imagine that r ceases to be active as an obstacle intersecting $L_x(t)$. If the left side of r is $[r_1, r_2]$, then $r_1 = r_3$ and $r_2 = r_4$ (of course, identifying points with their projections on the y axis), so that r_3 and r_4 already belong to $T_x(t')$ (as they were inserted at the left event when L_x touched the left side of r and, clearly, could not have been removed by any intermediate events).

Updating D: Delete r_3 and r_4 from $T_x(t')$, and assume the labels of the intervals just below r_3 and just above r_4 in $L_x(t')$ are (p, c) and (p', c'), respectively.

Updating T_x: Three cases may arise according as:

(i) r_4 is closer to p then p': Extend the upper endpoint of the interval just below r_3 to at least r_4, and then proceed to extend it further upwards exactly as in the manner for a point event.

(ii) r_3 is closer to p' then p: Extend the lower endpoint of the interval just above r_4 to at least r_3, and then proceed to extend it further downwards exactly as in the manner for a point event.

(iii) r_4 is closer to p' and r_3 is closer to p: Determine the point $r \in (r_3, r_4)$ which is equidistant from p and p', and extend the intervals just below r_3 and just above r_4 to meet at r.

See Figure 5.

Some useful facts that are not hard to verify, and help bound the complexity of maintaining the status tree T_x, are collected in:

Lemma 2 *The following hold:*

1. *The update procedure does indeed maintain T_x as well as the labels of its vertices correctly (i.e., according to conditions 1–4 for the labels B_i given at the beginning of the section).*
2. *If the labels (p, c) and (p', c) were both associated with vertices of T_x, even at different times, then $p = p'$, implying that the set of possible labels has cardinality $O(n + k)$.*
3. *No two vertices of T_x at a given instant can have the same label.*
4. *If a label B is associated with some vertex of T_x at time t', but is not associated with any vertex of T_x at a time $t > t'$ (i.e., the vertex associated with B has been deleted by some intermediate event), then B can never again be associated with any vertex of T_x at a time $t'' > t$. In other words, once a vertex with a given label is deleted that label cannot reappear.*
5. *At each event, the update of T_x involves at most three insertion or relabeling operations.*

Proof.

1: Follows from the description of the update procedures.

2: If (p, c) is a label consider two cases:

(a) $c \in S$, in which case we must have $p = c$, and,

(b) $c \in C_l$, in which case p is a closest to c of the points of S from which there is an x-path to c. However, after the choice of p is made, at the time of updating D at the left event when L_x lies on c, it is never changed.

In either case, we see that $(p, c) = (p', c) \Rightarrow p = p'$.

3: This follows from condition 4 for the labels B_i given at the beginning of the section.

4: Follows by examining each case where a deletion may occur in the update procedure.

For example, consider the case of a point event at q that results in the deletion of the vertex, say $a_2 \in T_x$ with label (p, c). This implies that the base $\bar{c}$ of the perpendicular from c to $L_x(t)$ is closer to q than p. See Figure 6.

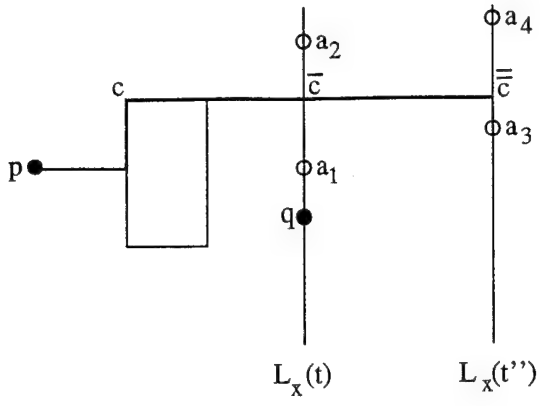

Figure 6: Illustration for the proof of Lemma 2.

If, at some time $t'' > t$, the label (p, c) reappears in T_x, say with vertex a_4, then we must have that the base $\bar{\bar{c}}$ of the perpendicular from c to $L_x(t'')$, which passes through $\bar{c}$, is closer to p than q. This is clearly not possible.
5: Follows by a scrutiny of each case of the update procedure. □

Lemma 2 implies:

Lemma 3 *T_x is of size $O(n+k)$ throughout, and the total number of operations performed on T_x through the sweep is $O(n+k)$. Therefore, the maintenance of T_x, which is height-balanced, through the sweep takes $O((n+k)\log(n+k))$ time.* □

At the end of the sweep we know, from the contents of D, for each point $q \in S$, a point p which is closest to q amongst points of S, different from q, from which there is an x-path to q.

We, therefore, have the following:

Proposition 4 *In $O((n+k)\log(n+k))$ time one can determine, following a sweep of the plane in the x direction, for each point $q \in S$, a point p which is closest to q amongst points of S, different from q, from which there is an x-path to q.*

In an exactly similar manner we can determine, within the same time bound, but following a sweep of the plane in the z direction (where $z = -x$, or $\pm y$), for each point $q \in S$, a point p which is closest to q amongst points of S, different from q, from which there is a z-path to q. □

Now, the following is easy to see:

Observation 1 *For any two points $p, q \in S$, there is either a shortest x-path or a shortest y-path from one of p, q to the other.*

Consequently, a closest pair can be determined by examining, for each point $q \in S$, points of S, different from q, which are closest to q with distances being measured to q either along x-paths or y-paths. Applying Proposition 4 we have:

Theorem 1 *In $O((n+k)\log(n+k))$ time one can determine, following sweeps of the plane in the x and y directions, a closest pair amongst points of S.* □

4 Finding Nearest Neighbors

Determination of a nearest neighbor for each point $q \in S$ is also straight-forward using Proposition 4. Relevant is:

Observation 2 *For any point $p \in S$, the shortest path from p to a given point $q \in S$ is a z-path, where z is either $\pm x$ or $\pm y$ (these are, of course, not mutually exclusive possibilities).*

Proposition 4, therefore, gives:

Theorem 2 *In $O((n+k)\log(n+k))$ time one can determine, following sweeps of the plane in each of the four axes-parallel directions, a nearest neighbor for each point $q \in S$.* □

5 Computing the Voronoi Diagram

5.1 The Plan

Denote the Voronoi diagram of a point set S in the presence of an obstacle set R of rectangles by $V_R(S)$. Due to the presence of obstacles, one needs to be careful about the structure of $V_R(S)$: it is a planar straight-line graph (PSLG) such that each face is either a rectangle of R, or corresponds to one point of S, say p, and is the locus of points ($\in G - \cup_{r \in R} interior(r)$) that are as close to p as to any other point of S. Call the face corresponding to p the V-cell of p, and denote it $V_R(S,p)$. The boundary of $V_R(S,p)$, denoted $B_R(S,p)$, is a polygonal line, consisting of one or more connected components, each point of which lies either

- on a bisector between p and some other $q \in S$, or
- on the boundary of an $r \in R$.

It may be observed that, due to the rectilinearity of the metric, each straight-line segment of a bisector is either horizontal, vertical, or inclined at 45° or 135° to the positive direction of the x-axis. See Figure 7 for illustration. To avoid cumbersome technicalities, we shall, in case of bisectors with non-zero area (see [12]) choose vertical lines as bisectors, and, further, make a standard assumption of "general position" such as no more than three points of S are co-circular.

The following bounds the complexity of $V_R(S)$, where $|R| = n$ and $|S| = k$:

Lemma 4 *The complexity of $V_R(S)$ as a planar graph is $O(n+k)$.*

Proof. Consider the planar graph $\overline{V}$ whose vertex set consists of
(a) Voronoi vertices (where three bisectors meet), and
(b) vertices where a bisector meets a rectangle,
and whose edge set consists of
(a) pieces of rectangle boundaries, and
(b) pieces of bisectors,
that join such vertices, ignoring the individual segments that comprise each piece.

See Figure 8(a) for the graph corresponding to the Voronoi diagram of Figure 7.

Then $\overline{V}$ is a planar graph with vertices of degree three (and no higher by the assumption of general position), so by Euler's formula the complexity of $\overline{V}$ is linear in the number of its faces which is, of course, at most $n+k$. Note that $\overline{V}$ may have parallel edges and edges going off to "infinity". Note also that rectangles that do not intersect bisectors are *unrepresented* in $\overline{V}$ (see Figure 8(b) and (c)).

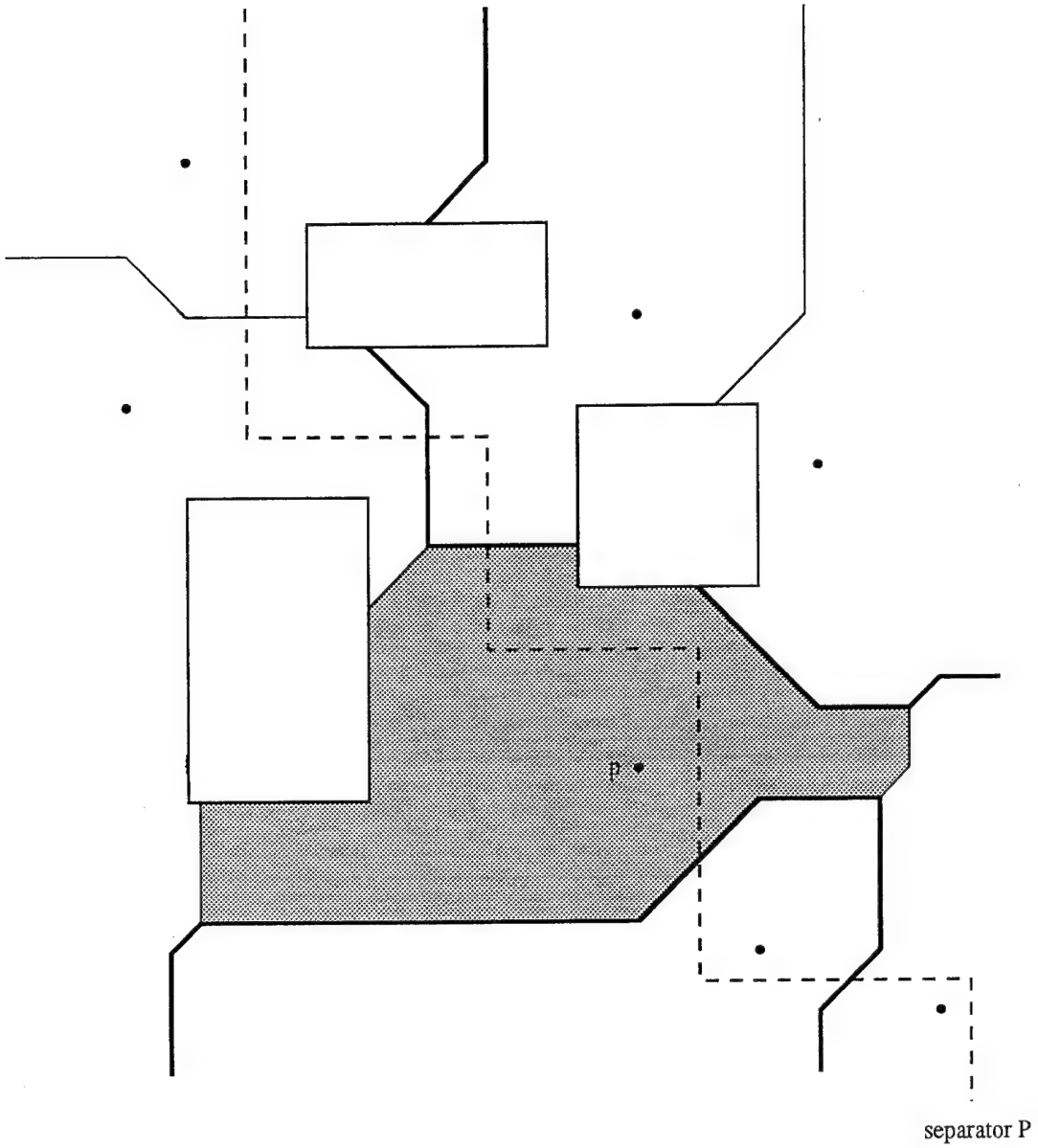

Figure 7: The Voronoi diagram $V_R(S)$ for a set S of seven points and a set R of three rectangles. The shaded region is the V-face $V_R(S, p)$ of p. The separator (dashed line) splits S into S_1 and S_2, and the bisector between them is shown in bold.

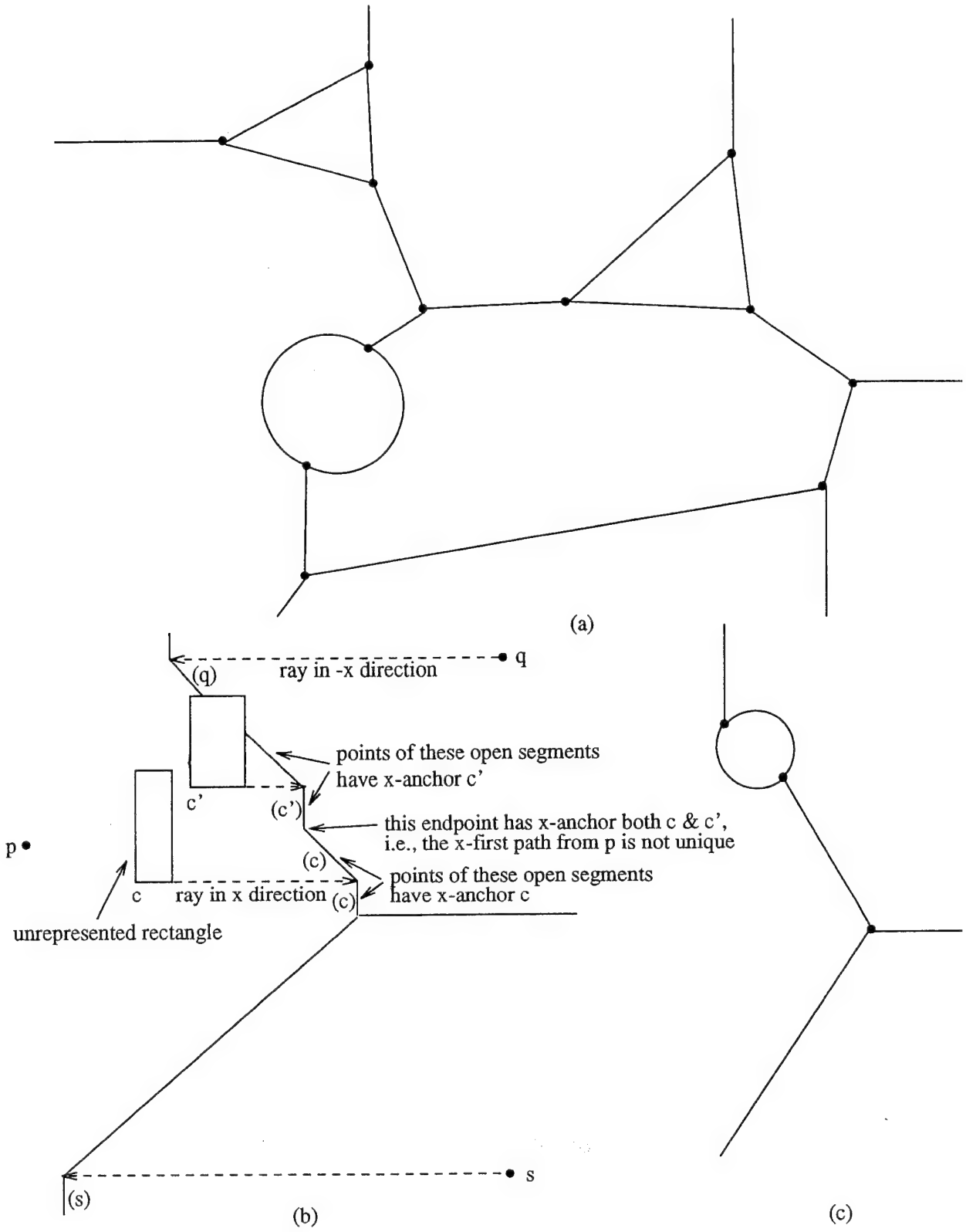

Figure 8: (a) The planar graph $\overline{V}$ corresponding to the Voronoi diagram in the previous figure. (b) A Voronoi diagram showing the points of $S \cup C$ (indicated in parenthesis) to which non-last edges of bisector pieces are charged. (c) The planar graph corresponding to the Voronoi diagram in (b).

Now, for the complexity of $V_R(S)$ we have to take into account, in addition to the complexity $\overline{V}$:

(1) the complexity of the unrepresented rectangles. This clearly totals $O(n)$.

(2) the complexity of the pieces of rectangle boundaries that correspond to edges of type (a). Each such piece comprises at most 4 segments, so the total complexity is again $O(n)$.

(3) the complexity of the pieces of bisectors that correspond to edges of type (b). This requires more careful analysis that we do next.

Analysis of case (3): We must count all segments comprising those pieces of bisectors that correspond to edges of type (b). For this we shall describe an accounting scheme where each such segment is "charged" either to a point in S, a corner of a rectangle in R, or an edge of $\overline{V}$.

Consider an arbitrary edge of $\overline{V}$ of type (b). Say it corresponds to the piece b of the bisector between points p and q of S. Orient b arbitrarily and imagine proceeding along b, segment by segment, from the first segment to the last. Assume each segment to be *open* (i.e., it does not include its endpoints). For each segment e of b, except the last, denote its successor by e'.

It is not hard to see that, as a consequence of the rectilinear geometry, exactly one of the following 8 cases must hold, where each case is obtained by replacing m by p or q, and z by $\pm x$ or $\pm y$, in the following statement:

> The shortest path from m to each point of $e(e')$ is a z-path and there is a unique z-first path to each point of $e(e')$, all with common z-anchor $c(c')$, *where*
>
> (a) either, $c \neq c'$,
>
> (b) or, $c = c'$, and the ray in the z direction from c intersects b at the endpoint between e and e'.
>
> (Note that the z-anchor of a segement may be a point of S or C, the set of corners of rectangles of R, and is unique as each segment is open.)

In this case, charge the segment e to the z-anchor c. See Figure 8(b).

It may also be seen that, conversely, if a point in $S \cup C$ does, in fact, accumulate a charge in this case, then m is uniquely determined as the point of S closest to c. And, c can accumulate at most 2 charges in each case corresponding to $z = \pm x$ or $\pm y$, from the two segments on either side of the endpoint where the ray in the z direction from c *possibly* intersects a bisector. Therefore, each point of $S \cup C$ may accumulate at most 8 charges under our accounting scheme. This proves that the number of segments that are not the last segments in the bisector pieces to which they belong is $O(n + k)$.

Finally, charge each last segment of a bisector piece to that edge of $\overline{V}$ to which the piece corresponds, so that each edge of $\overline{V}$ is charged once, proving that the number of such segments is also $O(n + k)$.

Thus, the number of all segments comprising those pieces of bisectors that correspond to edges of type (b) is $O(n + k)$, and this concludes our analysis of case (3).

Adding the complexities of cases (1), (2), and (3) to that of $\overline{V}$ proves the lemma. □

The overall plan for our Voronoi diagram algorithm is divide-and-conquer with an extension phase:

Apply Proposition 3 to find an $x(-y)$-path P such that the subsets of S and R to the left and right of P are S_1 and R_1, and S_2 and R_2, respectively. This guarantees that the number of rectangles in R_1 and R_2 is some constant fraction of the number of rectangles of R (note that we do not need any such assumption of "good" separation on S_1 and S_2 for our algorithm to work). Recursively compute $V_{R_1}(S_1)$ and $V_{R_2}(S_2)$. Then, *extend* $V_{R_i}(S_i)$ to $V_R(S_i)$, $i = 1, 2$, and, finally, *merge* $V_R(S_1)$ and $V_R(S_2)$ with a Shamos-Hoey type scan [16] to obtain $V_R(S)$.

5.2 Extending a Diagram

Let us consider the problem of extending $V_{R_1}(S_1)$ to $V_R(S_1)$ (extending $V_{R_2}(S_2)$ to $V_R(S_2)$ is exactly similar). If the closed region of the plane left (right) of P is denoted P_1 (P_2), it is clear that $V_R(S_1) \cap P_1 = V_{R_1}(S_1) \cap P_1$, as the shortest path joining two points in P_1 lies wholly in P_1, even considering all obstacles in R. Therefore, it remains to construct $V_R(S_1) \cap P_2$, that is extend $V_{R_1}(S_1)$ right of P.

Consider points $p \in S_1, q \in P_2$. Clearly, q can be situated in either the north-west, north-east, or south-east quadrants as viewed from p, and, given the shape of P, in the first case there can be no vertical blockage between them, and in the third case there can be no horizontal blockage. Recalling Proposition 1 we have:

Observation 3 *A shortest path from a point $p \in S_1$ to a point $q \in P_2$ is either an x-path or a y-path.*

Therefore, to find a $p \in S_1$ nearest to a given $q \in P_2$ (or, equivalently, the V-cell $V_R(S_1, p)$ containing q), we need only consider x-paths and y-paths from points of S_1 to q, suggesting, in fact, the geometric methods of Section 3.

This leads to the definition and construction of *z-diagrams*, for $z = \pm x, \pm y$. In particular, let us describe the x-diagram.

Constructing the x-diagram of S_1 and R: Sweep the plane G, containing only S_1 and R, as in Section 3, in the x direction. As the line L_x sweeps through the plane, build, on the fly, a planar straight-line graph, called the x-diagram of S_1 with obstacle set R, and denoted $X_R(S_1)$, such that each face of $X_R(S_1)$ is either

- a rectangle of R, or
- corresponds to one point of $S_1 \times (S_1 \cup C_l)$, say (p, c) (where C_l is the set of corners of left edges of rectangles of R; see the discussion of labels at the beginning of Section 3), in which case it is the locus of those points q such that, amongst the points of S_1 from which there is an x-path to q, p is the nearest and c is the x-anchor of q w.r.t. p, or
- the one remaining face, not of the preceding two types, that is the locus of those points $q \in G - \cup_{r \in R} \mathit{interior}(r)$ that cannot be reached from any point of S_1 by an x-path.

See Figure 9 for illustration. $X_R(S_1)$ can be built, and represented as a doubly-connected-edge-list (DCEL, see [15]), by tracing the motion on the plane of points of the status tree T_x (lying on L_x, of course: imagine an inkspot at each such point), and, when the sweep-line L_x touches a point $p \in S_1$ (or, $c \in C_l$) at time t, tracing the motion on the plane (along $L_x(t)$) as we proceed in either direction from p (or, possibly, c) to determine the extent of the new interval with label (p, p) (or, possibly, (p, c): imagine marking this new interval with a pen; see the discussion on updating the status tree in Section 3).

Observation 4 *Each face of $X_R(S_1)$ is a rectilinear polygon, and the face of $X_R(S_1)$ corresponding to $(p, c) \in S_1 \times (S_1 \cup C_l)$, called the x-*cell *of (p, c) and denoted $X_R(S_1, (p, c))$, contains c on its boundary. $X_R(S_1)$ is, of course, a planar graph analogous to a Voronoi diagram $V_R(S_1)$, but where distances are measured only along x-paths and that, further, has been refined up to anchors.*

Assume the number of points in S_i, $i = 1, 2$, is k_i. The following bounds the complexity of the x-diagram, as well as that of computing it:

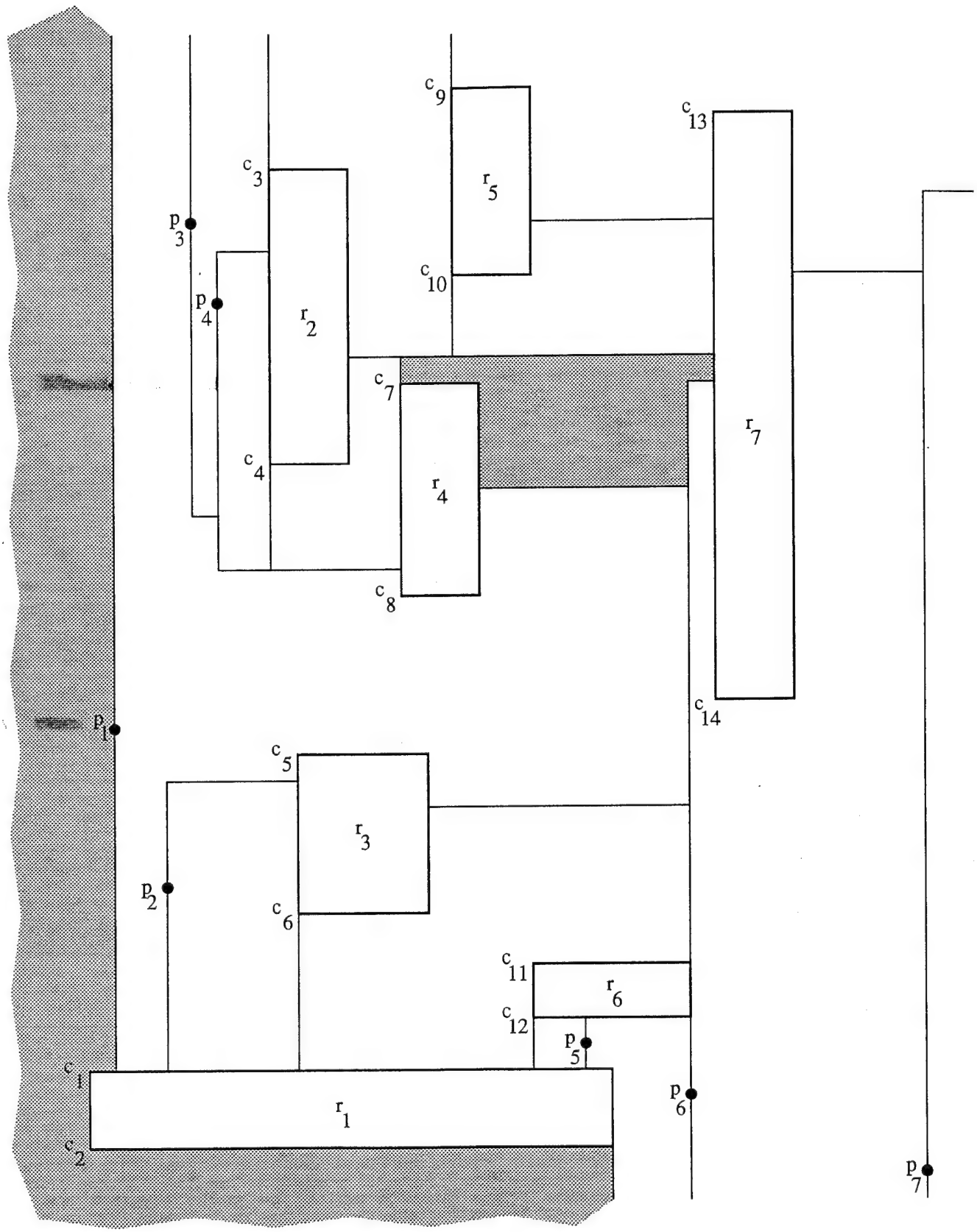

Figure 9: The x-diagram $X_R(S_1)$, for $S_1 = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$ and $R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}$. The shaded region on the left consists of points that cannot be reached from points of S_1 by x-paths, while the shaded region in the center, for example, is

Lemma 5 *The complexity of $X_R(S_1)$ as a planar graph is $O(n+k_1)$, and the time to compute it is $O((n+k_1)\log(n+k_1))$.*

Proof. The complexity of $X_R(S_1)$ as a planar graph follows from standard geometric arguments, while the time to compute it is dominated by the time to sweep the plane with L_x, which is $O((n+k_1)\log(n+k_1))$ from Lemma 3. □

This completes a description of constructing the x-diagram $X_R(S_1)$.

One can define and construct, in an exactly similar manner, the z-diagrams $Z_R(S_1)$, for $z = -x, \pm y$ ($Z = -X, \pm Y$, respectively). In particular, construct the y-diagram $Y_R(S_1)$.

Now note that, by Observation 3, $X_R(S_1) \cap P_2$ and $Y_R(S_1) \cap P_2$ together contain essentially all the proximity information of $V_R(S_1) \cap P_2$. How then do we use these two diagram to explicitly construct $V_R(S_1) \cap P_2$? Our plan is to scan the plane upwards with a horizontal sweep-line L_y, starting at a position sufficiently far south and constructing the part of $V_R(S_1) \cap P_2$ beneath L_y as we proceed.

Before describing this construction, we record certain relevant geometric features of $V_R(S_1) \cap P_2$.

Proposition 5 *The following hold:*

1. *Each edge e of $V_R(S_1) \cap P_2$ is a straight-line segment that is either horizontal, vertical, or inclined at 45° to the positive direction of the x-axis. In particular, no edge of $V_R(S_1) \cap P_2$ is inclined at 135° to the positive direction of the x-axis.*

2. *Each edge e of $V_R(S_1) \cap P_2$ is one of the following five mutually exclusive types:*

   - *P-edge: e is part of an edge of P.*
   - *R-edge: e is part of an edge of a rectangle of R within P_2.*
   - *XY-edge: e is part of the bisector between two points $p, q \in S_1$, such that the shortest path to any point of e from p is an x-path and the shortest path to any point of e from q is a y-path.*
   - *XX-edge: e is part of the bisector between two points $p, q \in S_1$, such that e is not of type XY and the shortest paths to any point of e from both p and q are x-paths.*
   - *YY-edge: e is part of the bisector between two points $p, q \in S_1$, such that e is not of type XY and the shortest paths to any point of e from both p and q are y-paths.*

3. *An inclined edge is always an XY-edge, a vertical edge may be either an XY- or YY-edge, while a horizontal edge may be either an XY- or XX-edge.*

4. *Each vertex v of $V_R(S_1) \cap P_2$ is one of the following six mutually exclusive types:*

   - *P-vertex: v is a corner of P.*
   - *R-vertex: v is a corner of a rectangle of R within P_2.*
   - *B-vertex: v is a point at which the bisector between two points of S_1 turns (from one straight-line segment to another) within P_2.*
   - *BP-vertex: v is the intersection with P of the bisector between two points of S_1.*
   - *BR-vertex: v is the intersection with an edge of a rectangle of R, within P_2, of the bisector between two points of S_1.*

- *V-vertex: v is the intersection of three bisectors within P_2 (i.e., a Voronoi vertex).*

Proof. The classification of the edges and vertices of $V_R(S_1) \cap P_2$ (items 2 and 4) follows immediately from the definition of $V_R(S_1)$ and Observation 3. (See Figure 10 for illustration.)

For the proof of item 1, observe that, given two points p_1 and p_2 such that a segment of the bisector between them is inclined at 135° to the positive direction of the x-axis, and given a subsegment d of that inclined segment, one of p_1 and p_2 must lie in the upper right "quadrant" determined by d (refer Figure 11(a)). Now, suppose that there exists an edge e of $V_R(S_1) \cap P_2$ inclined at 135° to the positive direction of the x-axis. This would then imply (refer Figure 11(b)) that a point of S_1 lies to the right of P, which is impossible.

For item 3, we can use an argument similar to the one given above to show that an XX-edge can run only horizontally, and a YY-edge only vertically. We leave the details to the reader. □

Observe that at any instant through the intended sweep, when L_y lies along $L_y(t)$, $L_y(t) \cap (V_R(S_1) \cap P_2)$ consists of zero or more horizontal edges of $V_R(S_1) \cap P_2$, and the intersections of some vertical and inclined edges of $V_R(S_1) \cap P_2$ with $L_y(t)$ (these intersections may, in fact, be vertices of $V_R(S_1) \cap P_2$). We keep a description of $L_y(t) \cap (V_R(S_1) \cap P_2)$ as a sorted sequence $\{x_i : i = 0, \ldots\}$ of points stored in records of a height-balanced status tree T_y. Each x_i is either the endpoint of a horizontal edge of $V_R(S_1) \cap P_2$, or the intersection of a vertical or inclined edge of $V_R(S_1) \cap P_2$ with $L_y(t)$, which information is stored at the record for x_i (of course, $x_0 = P \cap L_y(t)$).

Our plan is to maintain T_y (at least implicitly, but not necessarily exactly, which we justify later) through the sweep. This will allow us to build $V_R(S_1) \cap P_2$, and represent it as a DCEL, on the fly.

Initialize the event point schedule E by including in it the following points sorted by y-coordinate:

1. P-vertices of $V_R(S_1) \cap P_2$ that are left endpoints of horizontal edges of P.
2. R-vertices of $V_R(S_1) \cap P_2$ that are left endpoints of horizontal edges of $r \in R_2$.
3. BP-vertices of $V_R(S_1) \cap P_2$.
4. BR-vertices of $V_R(S_1) \cap P_2$ that lie on *upper* or *right* edges of rectangles of R_2.

Points of the third and fourth type are called *start* vertices as they mark "starting" endpoints for bisectors in $V_R(S_1) \cap P_2$.

It should be remarked that new event points, either so-called LE-vertices (to be defined) or V-vertices, will be added to E as the sweep proceeds. V-vertices are also start vertices.

We claim:

Lemma 6 *E can be initialized in $O((n + k_1)\log(n + k_1))$ time, and within that same time one can determine, for each start vertex v of E, the direction and type of the edge, say e, that starts at that vertex, as well as the identity of the bisector of which e is a part (i.e., the two points of S_1 that this bisector, in fact, bisects).*

Proof. It is trivial to include points of the first two types. Points of the third type can be determined as the intersections of edges of $V_{R_1}(S_1)$ (which has been recursively computed) with P.

For points of the fourth type, we claim that, in fact, all BR-vertices of $V_R(S_1) \cap P_2$ can be determined in $O((n + k_1)\log(n + k_1))$ time by the following steps:

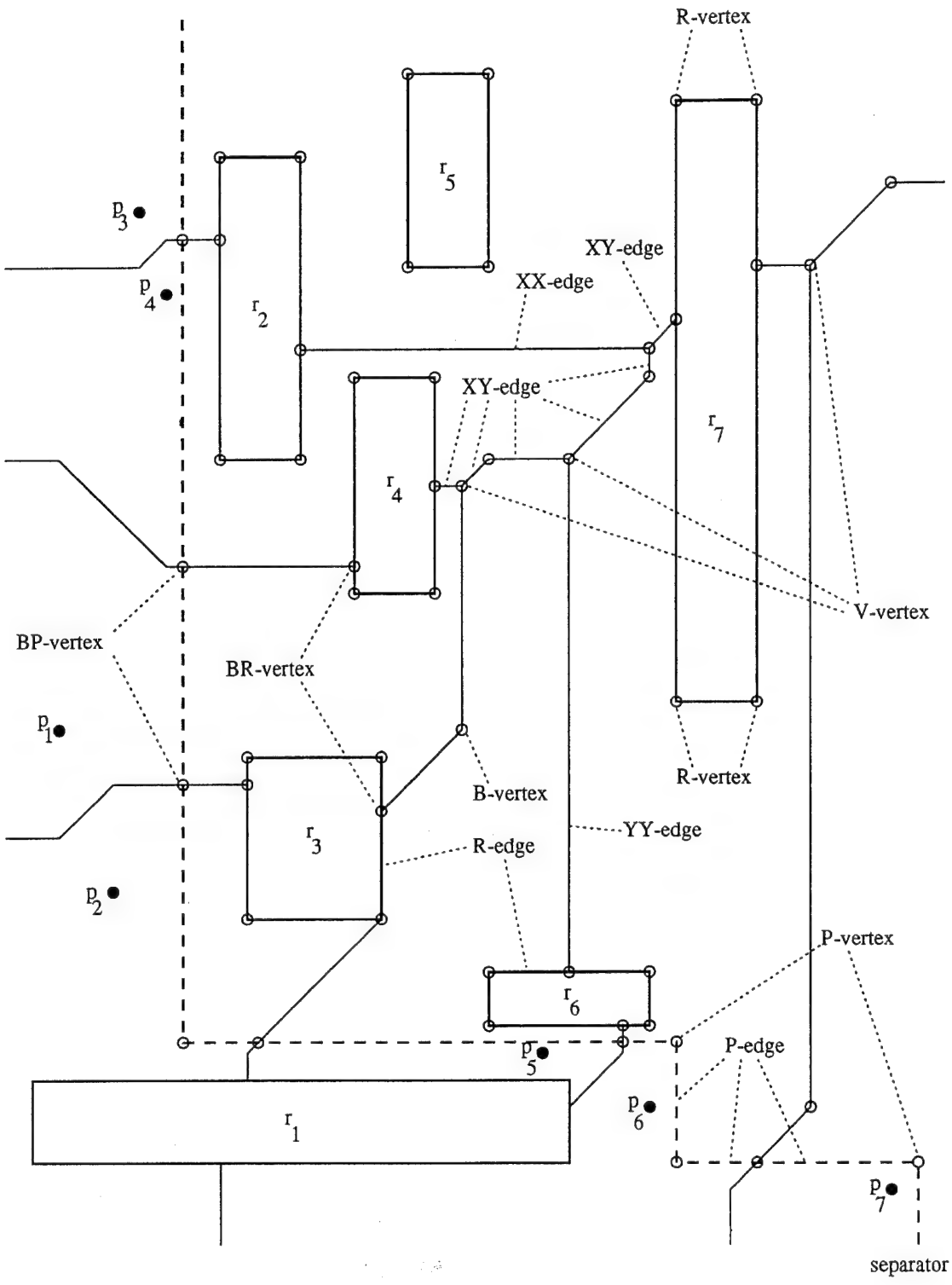

Figure 10: The hollow circles are the vertices of $V_R(S_1) \cap P_2$, and its edges include the dashed lines along the separator P as well as the solid lines to the right of P (for $S_1 = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$ and $R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}$ of Figure 9). *Some* of the vertices and edges have been labeled according to Proposition 5.

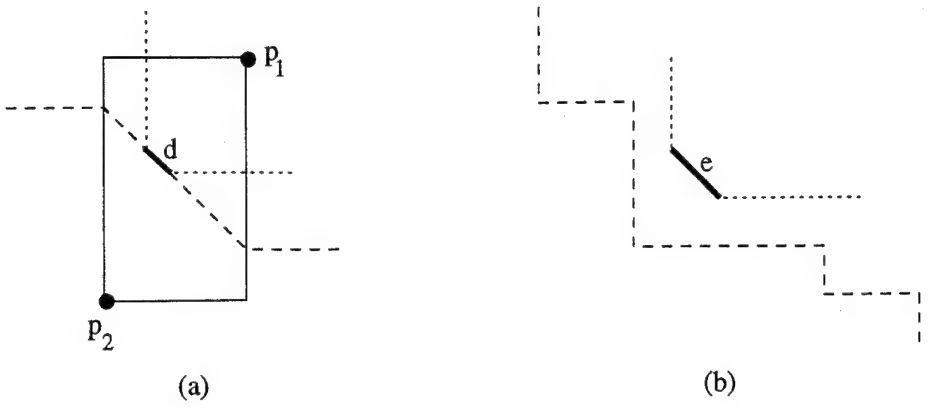

Figure 11: Illustration for the proof of Proposition 5: (a) The disposition of two points whose bisector has a segment inclined at 135° to the positive direction of the x-axis, and (b) part of such a bisector to the right of P.

(a) Preprocess the planar graphs $X_R(S_1)$ and $Y_R(S_1)$, for logarithmic time point location, in time $O((n+k_1)\log(n+k_1))$ (see [15]).

(b) Locate the lower left corner of each rectangle of R_2 in both graphs $X_R(S_1)$ and $Y_R(S_1)$ in total time $O(n\log(n+k_1))$.

(c) Traverse clockwise the boundary of each $r \in R_2$, starting from its lower left corner, while tracking, at the same time, the nearest points of S_1 and S_2. For this, by Observation 3, we simply need to determine the successive x-cells and y-cells that are crossed through the traversal. And, during the traversal, a point v on the boundary of r that is nearest to two distinct points of S_1 is marked as a BR-vertex. □

Before describing the rather complex procedure for updating T_y at each event point, let us try to provide the guiding geometric insight which is relatively simple.

View $V_R(S_1) \cap P_2$ as a "set of (connected) bisector-components", each bisector-component starting at either a BP-vertex or a BR-vertex (on the upper or right edge of a rectangle of R), and traveling monotonically in the upwards direction either to "infinity", or ending at a V-vertex or on the lower or left edge of a rectangle. This view is motivated by the first item of Proposition 5, and justifies the inclusion of the third and fourth types of points in the schedule E (of course, points of the first and second types are included in E as they mark the predictable events of including edges of P and rectangles of R into $V_R(S_1) \cap P_2$).

Now, a YY-bisector-component (i.e., a bisector-component consisting of YY-edges: it is not hard to see that if one edge of a bisector component is of type YY (XY, XX), then every edge of that bisector-component is of type YY (XY, XX)) travels vertically upwards from its start vertex, either indefinitely, or till it ends either at a V-vertex, or on the lower edge of a rectangle (the second case may be viewed as the YY-bisector-component being struck from the left by a horizontal edge of $V_R(S_1) \cap P_2$).

An XX-bisector-component travels horizontally rightwards from its start vertex, either indefinitely, or till it ends at a V-vertex, or on the left edge of a rectangle (the second case may be viewed as the XX-bisector-component striking a vertical edge of $V_R(S_1) \cap P_2$).

With an XY-bisector-component we must be more careful as it may make turns while traveling from its start vertex. Even so, the XY-bisector-component may be tracked with the help of the x- and y-diagrams, for within a fixed x-cell and a fixed y-cell (where the x-anchors and

y-anchors of points w.r.t. nearest points in S_1 do not change) the bisector-component makes at most two turns and its path may be determined easily. Thus, we track the XY-bisector-component first from its start vertex v, through the intersection of the x-cell and y-cell to which v belongs, to the point, say v_1, where it first leaves one of these two cells to enter a new x- or y-cell. Note that the XY-bisector-component must make a turn at v_1, and we repeat the tracking procedure starting from v_1, ..., either indefinitely, or till it ends either at a V-vertex or on the lower or left edge of a rectangle.

With this insight in mind, we shall describe the detailed procedure for updating T_y. However, before beginning the procedure we need a preprocessing stage:

Preprocess each x-cell of $X_R(S_1)$ and y-cell of $Y_R(S_1)$ for $O(\log(n+k_1))$-time *ray-shooting* (i.e., to determine where a ray fired in some given direction from some given point in that cell will first strike the boundary of the cell) in total $O(n+k_1)$ time (see [9] for linear-time preprocessing for ray-shooting). Further, preprocess the horizontal edges of both P and rectangles of R, again in total $O(n+k_1)$ time, for $O(\log(n+k_1))$-time *vertical ray-shooting* (i.e., determining which of these horizontal edges, if any, will be struck by a vertical ray fired from some given point on the plane, see the computation of *vertical adjacency maps* [15, p. 349]).

Procedure for updating T_y: Initially, when $L_y(t)$ lies south of any point in the schedule E, $T_y(t)$ contains only one point, the intersection of $L_y(t)$ with the lowest vertical edge of P. Next, start scanning upwards with L_y, stopping at successive event points of E to update T_y as follows:

1. At a P-vertex v, v becomes the new left endpoint of the intersection $L_y(t) \cap (V_R(S_1) \cap P_2)$. Insert v "appropriately" into $T_y(t)$ (this includes recording the horizontal and vertical edges of P in $V_R(S_1) \cap P_2$ that start at v), and delete the previous leftmost point, say v_1, of T_y (this represents that a vertical edge of $V_R(S_1) \cap P_2$ ends at v_1).

2. At an R-vertex that is on the lower edge of an $r \in R$, appropriately insert both endpoints of that **lower edge** into $T_y(t)$ (the lower edge itself, as well as the vertical edges starting at either endpoint of that edge are parts of $V_R(S_1) \cap P_2$).

   At an R-vertex that is on the upper edge of an $r \in R$, appropriately delete both endpoints of that edge from $T_y(t)$ (they both mark endpoints of verticals edges of $V_R(S_1) \cap P_2$, while the horizontal edge joining them is part of $V_R(S_1) \cap P_2$).

3. At a start vertex v (i.e., either a BP-, BR-, or V-vertex), insert it into $T_y(t)$ as the starting endpoint of an edge e of $V_R(S_1) \cap P_2$, and also record the direction and type of e, as well as the identity of the bisector of which e is a part (see Lemma 6).

   If e is a YY-edge, record the point, say v_1 (possibly $+\infty$), at which the ray along e (i.e., vertically upwards) first hits a rectangle of R, as the "likely endpoint" of e (preprocessing for vertical ray-shooting allows us to compute v_1 in time $O(\log(n+k_1))$), by inserting v_1 appropriately into E as an LE-vertex.

   If e is an XX-edge, it is horizontal and, in fact, lies wholly along $L_y(t)$ and ends at the next vertex (to the right), say v_1, on $T_y(t)$ which must correspond either (a) to a vertical edge lying on the left side a rectangle of $r \in R$, or (b) a vertical edge that is part of some bisector. In both cases the vertical edge ends at v_1 (so, also remove the LE-vertex corresponding to this vertical edge from E). In case (a), a "new" vertical edge continues up the left side of r starting from v_1. In case (b), insert v_1 into E as a new V-vertex which is the start of an XY-edge e' that is part of a bisector whose identity is also trivially determined.

If e is an XY-edge, extend the bisector-component b of which it is a part to the point, say v_1, where b first strikes the boundary of either the x-cell or y-cell to which v belongs (determining v_1 takes time $O(\log(n+k_1))$ as we need to follow the bisector through at most two turns and, possibly, perform ray-shooting). Insert v_1 into E as an LE-vertex, recording it as the "likely endpoint" of b.

4. At an LE-vertex v_1 that lies on the edge (either left or lower) of a rectangle, mark v_1 as the actual endpoint of the corresponding edge of a bisector-component of $V_R(S_1) \cap P_2$. At an LE-vertex v_1 where an XY-edge strikes the boundary of either an x-cell or a y-cell, again mark v_1 as the endpoint of the corresponding edge of a bisector-component b of $V_R(S_1) \cap P_2$. It is, in fact, a turn point of b. Therefore, as in the last paragraph of Step 3, further extend b to the point, say v_2, where it next strikes the boundary of either the x-cell or y-cell to which v_1 belongs (again, determining v_2 takes time $O(\log(n+k_1))$). Insert v_2 into E as an LE-vertex, recording it as the "likely endpoint" of b.

Comment. The points in $T_y(t)$ may not exactly represent the intersections, at each instant t, of $L_y(t)$ with $V_R(S_1) \cap P_2$, because of the inclined edges of $V_R(S_1) \cap P_2$. However, this does not compromise either the procedure for updating T_y or constructing the DCEL for $V_R(S_1) \cap P_2$, as through the sweep we do succeed in locating both endpoints of every edge of $V_R(S_1) \cap P_2$, i.e., we do not miss any intersections of edges even though they may be traveling at an incline between event points (in fact, each intersection will be discovered as an event point).

It may be checked that there will be at most $O(n+k_1)$ updates to T_y at a cost of $O(\log(n+k_1))$ per update (the easiest way to count updates is to see the correspondence between each update and a vertex of $V_R(S_1) \cap P_2$). The only relevant non-trivial observation here is that each $O(\log(n+k_1))$ cost to track an XY-bisector-component b from one LE-vertex v to another may be "charged" to v (note that b must turn at v, so that v is, indeed, a vertex of $V_R(S_1) \cap P_2$), implying that the total cost of tracking all XY-bisector-components is $O((n+k_1)\log(n+k_1))$. (A simple intuition, in fact, a motivation for the entire procedure, is that, to trace an XX- or YY-bisector-component we "pay" only a constant amount, while, for an XY-bisector-component, we "pay" only when it turns.) We have, therefore:

Proposition 6 *We can extend $V_{R_i}(S_i)$ to $V_R(S_i)$, $i = 1, 2$, in $O((n+k)\log(n+k))$ time.* □

5.3 Merging Two Diagrams

Now, consider the problem of finding $V_R(S)$, given $V_R(S_1)$ and $V_R(S_2)$. We intend a Shamos-Hoey type scan ([16], see also [15]) to compute the *bisector* of S_1 and S_2, denoted $b(S_1, S_2)$, consisting of those points q ($\in G - \cup_{r \in R} interior(r)$), such that if p_1 and p_2 are the nearest to q amongst points of S_1 and S_2, resp., then $d(q, p_1) = d(q, p_2)$. But first:

Lemma 7 *The following hold:*

1. *The bisector $b(S_1, S_2)$ is a subgraph of $V_R(S)$ and, therefore, has complexity $O(n+k)$.*
2. *It consists of components, each a polygonal line that, in either direction, is either unbounded or ends on the boundary of an $r \in R$.*
3. *It divides $G - \cup_{r \in R} r$ into two (not necessarily connected) subsets $b^-(S_1, S_2)$ and $b^+(S_1, S_2)$ to its left and right, resp., w.r.t. to some total orientation.*
4. $$V_R(S) = (V_R(S_1) \cap b^-(S_1, S_2)) \bigcup b(S_1, S_2) \bigcup (V_R(S_2) \cap b^+(S_1, S_2)) \bigcup \cup_{r \in R} r.$$

Proof. Follows by standard geometric arguments, but see Figure 7. □

For the computation of $b(S_1, S_2)$ it is observed that:

The bisector $b(S_1, S_2)$ may be found by a scan *exactly* similar to the Shamos-Hoey type scan for an Euclidean space without obstacles, *except*

(a) that $b(S_1, S_2)$ may have turns even within one given V-cell because of the nature of L_1-bisectors, and because it passes through regions with different anchors w.r.t. the closest points in S_1 or S_2,

(b) that such turns may be found by simultaneously tracking $b(S_1, S_2)$ through the z-diagrams $Z_R(S_i)$ ($Z = \pm X, \pm Y$, $i = 1, 2$), as the z-cells to which a point $p \in b(S_1, S_2)$ belongs determine its anchors w.r.t. the closest points in S_1 and S_2, and, therefore, allow $O(\log(n+k))$ time determination of the straight segment of $b(S_1, S_2)$ on which p lies; of course, this requires $O((n+k)\log(n+k))$ time preprocessing of the z-diagrams, see the analogous procedure for tracking an XY-bisector-component in the procedure for updating T_y in the previous section, and,

(c) that the number of such turns is $O(n+k)$.

We have, therefore:

Proposition 7 *We can merge $V_R(S_1)$ and $V_R(S_2)$ to obtain $V_R(S)$ in $O((n+k)\log(n+k))$ time.* □

5.4 Putting Everything Together

We now have all the pieces required to execute the plan, described in Section 5.1, for an algorithm computing the Voronoi diagram:

Given the problem of computing the Voronoi diagram defined by S and R, where $|S| = k$ and $|R| = n$, we first subdivide (using Proposition 3) into two subproblems defined by S_1 and R_1, and S_2 and R_2, resp., where $|S_1| = k_1$, $|R_1| = n_1$, $|S_2| = k_2$, $|R_2| = n_2$, and $n_1, n_2 \geq \frac{1}{8}n$. This subdivision requires $O((n+k)\log n)$ time, as it takes $O(n \log n)$ time to build the the subdividing path P and, subsequently, $O(\log n)$ time per element of $S \cup R$ to determine on which side of P it lies by a binary search over the $O(n)$-sized P.

After recursively finding the diagrams that solve these two subproblems, we *extend* and then *merge* these two diagrams (using Propositions 6 and 7, resp.) to obtain a solution to the original problem. These two phases require a total time of $O((n+k)\log(n+k))$.

Observing that all the base cases of the recursion can be solved in a total time of $O(k \log k)$ (each base case is, in fact, defined by some number, say k_1, of points of S and *zero or one* rectangles of R, and can be solved in $O(k_1 \log k_1)$ time by a modification of the algorithm of [12]), it may be checked that a recurrence bounds the running time of the entire algorithm to $O((n+k)\log(n+k)\log n)$.

We have, therefore:

Theorem 3 *In $O((n+k)\log(n+k)\log n)$ time one can determine the Voronoi diagram of S.* □

6 Computing a Minimal Spanning Tree

The problem of computing a minimal spanning tree of S (in the presence of the obstacle set R) may, in fact, be reduced in linear time to that of computing the Voronoi diagram $V_R(S)$. The relevant observation is (see [15] for the insight):

Observation 5 *Computing a minimal spanning tree of S is equivalent to computing the minimal spanning tree of the dual graph of $V_R(S)$, where the weight of an edge joining the two vertices that are dual to the faces $V_R(S,p)$ and $V_R(S,q)$, resp., of $V_R(S)$, is $d(p,q)$.*

Since the dual graph of $V_R(S)$ is a planar graph of size $O(n+k)$ (and may also be determined from $V_R(S)$ in time $O(n+k)$), the algorithm of Cheriton-Tarjan [4] finds a minimal spanning tree in $O(n+k)$ time. This completes the linear time reduction of the minimal spanning tree problem to the Voronoi diagram problem, and gives:

Theorem 4 *In $O((n+k)\log(n+k)\log n)$ time one can determine a minimal spanning tree of S.* □

7 Conclusions

For each of the four problems (a)–(d) described in the introduction, reduction from the element uniqueness problem [7] provides an $\Omega(k\log k)$ time lower bound, while an $\Omega(n+k)$ time lower bound holds trivially because of the input size. This gives the $\Omega(n+k\log k)$ time lower bound which, to our knowledge, is the best available, and shows that each of our four algorithms is near-optimal. Whether they can be improved is, of course, open.

However, it seems unlikely that the time will be lowered below $O((n+k)\log(n+k))$ as, in some sense, each of the four problems has an intrinsic complexity of $\Omega((n+k)\log(n+k))$: for, if a part of each problem were to determine the validity of the input sets (i.e., all rectangles are disjoint and all points lie outside the rectangles) then, indeed, a reduction from element uniqueness (by considering point-sized rectangles) implies an $\Omega((n+k)\log(n+k))$ time complexity.

We mention that one apparently feasible approach to obtaining $O((n+k)\log(n+k))$ time complexity – avoid divide-and-conquer by first computing all four z-diagrams $Z_R(S)$, $Z = \pm X, \pm Y$, and then construct the entire Voronoi diagram $V_R(S)$ with one plane-sweep similar to that in Section 5.2 – runs into difficulties that we believe cannot be overcome in $O((n+k)\log(n+k))$ time. For, recall that, to successfully complete the sweep of Section 5.2 in $O((n+k_1)\log(n+k_1))$ time, it was crucial that an XX- (YY-, resp.) bisector-component simply runs horizontally (vertically, resp.) until either it hits a rectangle or is hit from below (left, resp.) by another Voronoi edge. Hence, processing (at $O(1)$ cost) for a bisector-component when it enters a new x- or y-cell was necessary *only* if it was an XY-bisector-component (and an XY-bisector-component *does*, in fact, make a turn at every such entrance, so "justifying" the $O(1)$ cost). However, it seems an analogous property does not hold if we try to construct the entire Voronoi diagram in a single sweep using the four z-diagrams, and it is not clear how to avoid expending $O(1)$ processing cost *each* time *any* bisector-component enters a new x- or y-cell.

Other directions to consider include extending or modifying the techniques described here to deal with more general classes of obstacles and different metrics. For example, we believe that similar techniques will work with convex polygonal obstacles, as well as "fixed orientation metrics" [17].

Acknowledgement: We thank the anonymous referees for several suggestions that improved the paper, and for pointing out a critical reference.

References

[1] B. Aronov, On the geodesic Voronoi diagram of point sites in a simple polygon, *Algorithmica* 4 (1989), 109-140.

[2] M. J. Atallah, D. Chen, Parallel rectilinear shortest paths with rectangular obstacles, *Proc. ACM Symp. on Parallel Algorithms and Architectures*, 1990, 270-279; *Computational Geometry: Theory and Applications* **1** (1991), 79-113.

[3] J. L. Bentley, M. I. Shamos, Divide-and-conquer in multidimensional space, *Proc. ACM Symp. on Theory of Computing*, 1976, 220-230.

[4] D. Cheriton, R. E. Tarjan, Finding minimum spanning trees, *SIAM J. Computing* **5** (1976), 724-742.

[5] K. L. Clarkson, S. Kapoor, P. M. Vaidya, Rectilinear shortest paths through polygonal obstacles in $O(n(\log n)^2)$ time, *Proc. ACM Symp. on Computational Geometry*, 1987, 251-257.

[6] P. J. de Rezende, D. T. Lee, Y. F. Wu, Rectilinear shortest paths with rectangular barriers, *Proc. ACM Symp. on Computational Geometry*, 1985, 204-213; *Discrete and Computational Geometry* **4** (1989), 41-53.

[7] D. Dobkin, R. Lipton, On the complexity of computations under varying sets of primitives, *J. Computer and Systems Sciences* **18** (1979), 86-91.

[8] S. Guha, *Parallel Algorithms for Polygonal and Rectilinear Geometry*, Ph.D. Thesis, Computer Science and Engineering, University of Michigan, Ann Arbor, 1991.

[9] L. Guibas, J. Hershberger, D. Leven, M. Sharir, R. Tarjan, Linear-time algorithms for visibility and shortest path problems inside a triangulated simple polygons, *Algorithmica* **2** (1987), 209-233.

[10] D. G. Kirkpatrick, Optimal search in planar subdivision, *SIAM J. Computing* **12** (1983), 28-35.

[11] R. C. Larson, V. O. Li, Finding minimum rectilinear distance paths in the presence of barriers, *Networks* **11** (1981), 285-304.

[12] D. T. Lee, Two-dimensional Voronoi diagrams in the L_p-metric, *J. ACM* **27** (1980), 604-618.

[13] D. T. Lee, C. D. Yang, T. H. Chen, Shortest rectilinear paths among weighted obstacles, *International J. Computational Geometry & Applications* **1** (1991), 109-204.

[14] J. S. B. Mitchell, L_1 shortest paths among polygonal obstacles in the plane, *Algorithmica* **8** (1992), 55-88.

[15] F. P. Preparata, M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, 1985.

[16] M. I. Shamos, D. Hoey, Closest-point problems, *Proc. IEEE Symp. on Foundations of Computer Science*, 1975, 151-162.

[17] P. Widmayer, Y. F. Wu, C. K. Wong, On some distance problems in fixed orientations, *SIAM J. Computing* **16** (1987), 728-746.

[18] Y. F. Wu, P. Widmayer, M. D. F. Schlag, C. K. Wong, Rectilinear shortest paths and minimum spanning trees in the presence of rectilinear obstacles, *IEEE Trans. Computers* **C-36** (1987), 321-331.

[19] C. D. Yang, D. T. Lee, C. K. Wong, On bends and lengths of rectilinear paths: a graph-theoretic approach, *International J. Computational Geometry & Applications* **2** (1992), 61-74.